Formalising a Model of the λ-calculus in HOL-ST

Sten Agerholm

October 31, 1994

Abstract

Most new theorem provers implement strong and complicated type theories which eliminate some of the limitations of simple type theories such as the HOL logic. A more accessible alternative might be to use a combination of set theory and simple type theory as in HOL-ST which is a version of the HOL system supporting a ZF-like set theory in addition to higher order logic. This paper presents a case study on the use of HOL-ST to build a model of the λ-calculus by formalising the inverse limit construction of domain theory. This construction is not possible in the HOL system itself, or in simple type theories in general.

1 Introduction

The HOL system [GM93] supports a simple and accessible yet very powerful logic, called higher order logic or simple type theory. This is probably a main reason why it has one of the largest user communities of any theorem prover today. However, it is heard every now and then that users cannot quite do what they would like to do, e.g. due to restrictions in the type system of higher order logic. Some 'hack' their way around the problems with great inconvenience and some move to a stronger type theory, or simply give up. A problem with stronger type theories is that they are not at all simple and easy to learn and there are quite a lot of different ones to choose among. On the other hand, set theory is also more powerful than simple type theory. It was invented many years ago, is well-established and is intuitive to most people. Hence, set theory might be useful for theorem proving.

As an experiment, Mike Gordon is currently developing a system, called HOL-ST, which supports both a ZF-like set theory and higher order logic [Gor94]. Combining the expressive power of set theory with the benefits of higher order logic, such as e.g. type checking, HOL-ST might provide a simple alternative to systems based on type theories, and a more powerful alternative to the HOL system as well.

In this paper we present a case study on the use of HOL-ST with an application taken from domain theory. This work was motivated by previous work on formalising domain theory in HOL [Age93, Age94a, Age94b] where it became clear that the inverse limit construction of solutions to recursive domain equations cannot be formalised in HOL easily. The type system of higher order logic is not rich enough to represent solutions directly though some difficult hacking might make their formalisation possible indirectly.

This paper shows that this non-trivial construction can be formalised in HOL-ST directly. The crucial difference between HOL and HOL-ST is that HOL-ST supports general dependent products of the form

$$\prod_{x \in X} Y(x)$$

whereas HOL does not.

The inverse limit construction can be used to give solutions to any recursive domain (isomorphism) equation of the form

$$D \cong \mathcal{F}(D)$$

where $\mathcal{F}$ is an operator on domains like sum, product and (continuous) function space, or any combination of these. Thus, the right tools based on the present formalisation might allow very general recursive datatype definitions in HOL-ST.

As an illustration of this, we present the construction of a domain D_∞ which satisfies the recursive domain equation

$$D_\infty \cong [D_\infty \to D_\infty]$$

where $[D \to E]$ denotes the domain of all continuous function from D to E. Hence, D_∞ provides a model of the untyped λ-calculus. In fact, we have derived a parametrised class of solutions to the above equation. Any domain (with an appropriate embedding) can be used as the initial element of a certain "chain" of domains (cpos), obtained by iterating the function space. The inverse limit construction provides a kind of "least upper bound", an inverse limit, of any such chain.

The present formalisation of the inverse limit construction employs the categorical method using embedding projection pairs, see e.g. [SP82, Plo83]. This was suggested by Plotkin as a generalisation of Scott's original inverse limit construction of a model of the λ-calculus in the late 60's. The formalisation is based on Paulson's accessible presentation in the book [Pau87] but Plotkin's [Plo83] was also used in part (in fact, Paulson based his presentation on this).

The paper is organised as follows. In Section 2 we provide an overview of set theory in HOL-ST. Section 3 introduces the formalisation of the basic concepts of domain theory and Section 4 presents the formalisation of the inverse limit construction. The continuous function space is considered as an operator for recursive domain equations in Section 5. It is proved to be a continuous covariant functor. This allows a class of models and in turn a concrete model of the λ-calculus to be constructed in Section 6. Section 7 contains some concluding remarks.

2 Set theory

One good thing about set theory is that most people know it. From our early years of education we are taught to think in terms of sets and we all have an intuitive understanding of set theoretic notions such as set membership, union, intersection and so on. This makes it easy to introduce set theory in a paper like this because, in fact, it is not necessary to do it, and therefore I won't. What I will do, however, is to say a little bit about how set theory technically was made available in HOL, the right paper to read about this is Gordon's [Gor94], and introduce some of the less well-known concepts of set theory as well. (Section 2.3, partly, and Section 2.6 present a few extensions of HOL-ST made by the present author.)

2.1 Basic concepts

HOL is extended with set theory by declaring a new type V and a new constant $\in: V \times V \to bool$ and then postulating eight new axioms about V and $\in$. The exact shape of these are not important here. The axioms justify the claim that terms of type V can be interpreted as sets and $\in$ can be interpreted as the membership predicate on such sets. They also allow new constants to be defined (or specified) such as $\emptyset$ for the empty set, $\cup$ for the union of two sets, $\bigcup$ for the union of a set of sets, $\mathbb{P}$ for power set, and so on. Sets can also be written using standard notation like

$$\{x_1, x_2, \ldots, x_n\} \text{ and } \{x \in X \mid P[x]\}$$

which denote a finite set (of sets) and a subset of a set specified by a predicate. Furthermore, sets can be constructed by taking the image of a HOL function on some set:

$$y \in \mathsf{Image}\, f\, X \stackrel{\text{def}}{=} \exists x.\ x \in X \land y = f\, x.$$

Here, f has type $V \to V$. Generally speaking, new sets must be constructed from existing sets some way, in principle starting from the empty set and then using the axioms.

2.2 Pairs and functions

Set theory is not built with standard notions such as pairs and functions. These must be derived. For instance, the ordered pair of two sets is defined as follows

$$\langle x, y\rangle \stackrel{\text{def}}{=} \{\{x\}, \{x, y\}\}$$

which allows the characteristic property of pairs to be proved:

$$\forall x_1 x_2 y_1 y_2.\ (\langle x_1, y_1\rangle = \langle x_2, y_2\rangle) = x_1 = x_2 \land y_1 = y_2.$$

The cartesian product of two sets X and Y can then be defined as the set of all pairs of elements in X and Y, respectively:

$$X \times Y \stackrel{\text{def}}{=} \{\langle x, y\rangle \in \mathbb{P}(\mathbb{P}(X \cup Y)) \mid x \in X \land y \in Y\}.$$

This uses the power set twice to yield sets of sets of elements of X and Y, a subset of which contains pairs only (by the definition of pairs).

We can take the power set once more and get the set of all relations between two sets:

$$X \leftrightarrow Y \stackrel{\text{def}}{=} \mathbb{P}(X \times Y).$$

We shall not be interested in such relations directly but note that certain relations can be interpreted as functions. These are called *set* functions when we want to distinguish them from *logical* functions of higher order logic. Set functions are single-valued relations:

$$X \nrightarrow Y \stackrel{\text{def}}{=} \{f \in X \leftrightarrow Y \mid \forall x y_1 y_2.\ \langle x, y_1\rangle \in f \land \langle x, y_2\rangle \in f \Rightarrow y_1 = y_2\}.$$

This set contains both partial and total functions. Here, we shall only use the subset of total functions:

$$X \to Y \stackrel{\text{def}}{=} \{f \in X \nrightarrow Y \mid \forall x \in X.\ \exists y \in Y.\ \langle x, y\rangle \in f\}.$$

Note that a set function (total or not) has HOL type V since it is a set. We use the notation $\forall xy \in A.\ t$ to abbreviate the term $\forall x.\ x \in A \Rightarrow \forall y.\ y \in A \Rightarrow t$. A similar notation is used for existential quantification and the choice operator.

2.3 Identity and composition

The two most basic functions, the identity function and the composition function, can be defined in set theory. Of course, the identity function is the simplest of the two

$$\mathsf{Id}\, X \stackrel{\text{def}}{=} \{\langle x, y\rangle \in X \times X \mid x = y\}$$

and it is not difficult to prove that it is a total set function:

$$\forall X.\ \mathsf{Id}\, X \in X \to X.$$

Note that it is not possible to define a set for Id without the set argument since sets must be build from existing ones.

The composition function could be defined in the same way, yielding a composition function which takes three sets and two functions as arguments:

$$\mathsf{ComposeFun}(X, Y, Z) f\, g.$$

This would be rather inconvenient to work with due to the set arguments so fortunately we are able to exploit the presence of the function arguments to define a simpler composition function:

$$f \mathbin{\mathsf{O}} g \stackrel{\text{def}}{=} \{\langle x, z\rangle \in \mathsf{domain}\, g \times \mathsf{range}\, f \mid \exists y.\ \langle x, y\rangle \in g \wedge \langle y, z\rangle \in f\}.$$

Here, domain and range are defined using Image by taking the first and second components of each pair in a set function. Note that the composition function is a logical function, it has type $V \to V \to V$. If we needed a composition function which was a set function then we would have to make use of the set arguments.

2.4 Function abstraction and application

We can introduce a λ-abstraction to write functions of set theory. Roughly speaking, function abstraction is defined as follows

$$(\lambda x \in X.\, f\, x) \stackrel{\text{def}}{=} \{\langle x, y\rangle \in X \times \mathsf{Image} f\, X \mid y = f\, x\}$$

where f is a logical function of type $V \to V$. Indeed, this yields a total set function:

$$\forall f X Y.\ (\forall x.\ x \in X \Rightarrow f\, x \in Y) \Rightarrow (\lambda x \in X.\, f\, x) \in X \to Y.$$

Note that in general the body of the abstraction can be any term, not only applications of the form $f\,x$.

A set function can be applied to an argument by using a special function application, written as $\diamond$. HOL function application by juxtaposition does not work since a set function has type V. The new function application is defined by:

$$f \diamond x \stackrel{\text{def}}{=} \varepsilon y.\ \langle x, y\rangle \in f.$$

If f is a set function in $X \to Y$ and $x \in X$ then $f \diamond x$ is an element of Y. Furthermore, we have

$$\forall X f x.\ x \in X \Rightarrow (\lambda x \in X.\, f\,x) \diamond x = f\,x$$

so function abstraction works as desired.

2.5 Natural numbers

The natural numbers can be represented in set theory by the set

$$\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \ldots\}.$$

This set is called Num. The number zero is the empty set and the successor is defined by

$$\mathsf{Suc}\, n \stackrel{\text{def}}{=} n \cup \{n\}.$$

The set Num represents the type of natural numbers *num* in a precise sense: there is a function $\mathsf{num2Num}$ of type $num \to V$ which is a bijection on Num, with inverse $\mathsf{Num2num}$ of type $V \to num$. These allow us to translate theorems about HOL numbers to theorems about numbers in set theory. The latter usually have assumptions of the form $n \in \mathsf{Num}$, as in

$$\forall n.\ n \in \mathsf{Num} \Rightarrow n \downarrow\!\leq \mathsf{Suc}\, n,$$

because $\mathsf{num2Num}$ is a bijection on Num only (not on all sets). Above we have used the arrow in the term $\downarrow\!\leq$ to say that the ordering relation is a version of the HOL relation $\leq$ in set theory.

2.6 Dependent sum and product

Until this point the HOL user is perhaps quite unimpressed by set theory which may seem only to make things more complicated. Hence, here are some goodies!

Many HOL users have been in a situation where they would have liked to have dependent types. Regarding ordinary HOL predicate sets as types it is possible to solve simple problems of this sort but more general constructions like dependent sum and dependent product are impossible to define, even using predicate sets. They are straightforward to define in set theory.

The dependent sum is a generalisation of the relation set introduced above where we defined $X \leftrightarrow Y$ to be the set of all relations between X and Y. In the dependent sum, the second set may depend on the first set:

$$\sum_{x \in X} Y\,x \stackrel{\text{def}}{=} \bigcup_{x \in X} \bigcup_{y \in Y\,x} \{\langle x, y \rangle\}.$$

Here, $Y\,x$ can be any term of type V containing x or not. In the same way, the dependent product is a generalisation of the total function set introduced above and written as $X \to Y$:

$$\prod_{x \in X} Y\,x \stackrel{\text{def}}{=} \{f \in \sum_{x \in X} Y\,x \mid \forall x \in X.\ \exists! y.\ \langle x, y \rangle \in f\}.$$

Clearly, both the relation set and the function set could have been obtained as special cases of the dependent sum and the dependent product, respectively. In fact, the function abstraction we introduced above yields an element of the dependent product:

$$\forall f X Y.\ (\forall x.\ x \in X \Rightarrow f\,x \in Y\,x) \Rightarrow (\lambda x \in X.\, f\,x) \in \prod_{x \in X} Y\,x.$$

In the present application of HOL-ST we shall make essential use of the dependent product to overcome limitations in the type system of the HOL logic.

3 Basic concepts of domain theory

Domain theory is the study of complete partial orders (cpos) and continuous functions between cpos. A pair consisting of a set and an ordering relation is called a complete partial order when it is a partial order and contains least upper bounds of all chains. A continuous function is a monotonic function (w.r.t. the relation) which preserves such least upper bounds. This section very briefly introduces the semantic definitions of the central concepts of domain theory in HOL-ST.

3.1 Complete partial orders

A partial order (po) is a HOL pair consisting of a set and a binary relation such that the relation is reflexive, transitive and antisymmetric on all elements of the set:

$$
\begin{aligned}
&\mathsf{po}\, D \stackrel{\text{def}}{=} \\
&\qquad \forall x \in \mathsf{set}\, D.\ \mathsf{rel}\, D\, x\, x \,\wedge \\
&\qquad \forall xyz \in \mathsf{set}\, D.\ \mathsf{rel}\, D\, x\, y \wedge \mathsf{rel}\, D\, y\, z \Rightarrow \mathsf{rel}\, D\, x\, z \,\wedge \\
&\qquad \forall xy \in \mathsf{set}\, D.\ \mathsf{rel}\, D\, x\, y \wedge \mathsf{rel}\, D\, y\, x \Rightarrow x = y.
\end{aligned}
$$

The constants set and rel equal FST and SND respectively. The constant po has type $V\#(V \to V \to \mathit{bool}) \to \mathit{bool}$.

We often confuse a (complete) partial order with the underlying set. Hence, we say a term is an element of a (c)po when we should really say that it is an element of the underlying set. The constants rel and set support this confusion.

A chain of elements of a partial order is a non-decreasing sequence of type $\mathit{num} \to V$:

$$\mathsf{chain}\, D\, X \stackrel{\text{def}}{=} (\forall n.\ X\, n \in \mathsf{set}\, D) \wedge (\forall n.\ \mathsf{rel}\, D(X\, n)(X(n+1))).$$

An upper bound of such a chain is an element of the po which is approximated by all elements of the chain:

$$\mathsf{isub}\, D\, X\, x \stackrel{\text{def}}{=} x \in \mathsf{set}\, D \wedge \forall n.\ \mathsf{rel}\, D(X\, n)x.$$

We are interested in the least such element, called the least upper bound (lub):

$$\mathsf{islub}\, D\, X\, x \stackrel{\text{def}}{=} \mathsf{isub}\, D\, X\, x \wedge \forall y.\ \mathsf{isub}\, D\, X\, y \Rightarrow \mathsf{rel}\, D\, x\, y.$$

When a lub exists we can use the choice operator to select this lub:

$$\mathsf{lub}\, D\, X \stackrel{\text{def}}{=} \varepsilon x.\ \mathsf{islub}\, D\, X\, x.$$

A lub is unique by antisymmetry of partial orders.

A partial order in which all chains have least upper bounds is called a complete partial order:

$$\mathsf{cpo}\, D \stackrel{\text{def}}{=} \mathsf{po}\, D \wedge \forall X.\ \mathsf{chain}\, D\, X \Rightarrow \exists x.\ \mathsf{islub}\, D\, X\, x.$$

From the definition of complete partial order we can prove the theorem

$$\forall D X.\ \mathsf{chain}\, D\, X \Rightarrow \mathsf{cpo}\, D \Rightarrow \mathsf{islub}\, D\, X (\mathsf{lub}\, D\, X)$$

which says that for any cpo and chain of elements in the cpo, the constant lub gives the least upper bound.

We do not require cpos to have a least element, also called a bottom (or an undefined) element. A cpo is called pointed if it has a least element:

$$\mathsf{pcpo}\, D \stackrel{\text{def}}{=} \mathsf{cpo}\, D \land \exists x \in \mathsf{set}\, D.\ \forall y \in \mathsf{set} D.\ \mathsf{rel}\, D\, x\, y.$$

The choice operator can be used to select a bottom element:

$$\mathsf{bot}\, D \stackrel{\text{def}}{=} \varepsilon x \in \mathsf{set}\, D.\ \forall y \in \mathsf{set}\, D.\ \mathsf{rel}\, D\, x\, y.$$

The constant bot works for any pointed cpo:

$$\forall D.\ \mathsf{pcpo}\, D \Rightarrow \forall y \in \mathsf{set}\, D.\ \mathsf{rel}\, D (\mathsf{bot}\, D) y.$$

By antisymmetry, the bottom is unique if it exists.

3.2 Continuous functions

So, cpos are partial orders which contains the lubs of all chains. We define the set of continuous functions as the subset of monotonic functions which preserve lubs of chains:

$$\begin{aligned}
&\mathsf{mono}(D, E) \stackrel{\text{def}}{=} \\
&\quad \{f \in \mathsf{set}\, D \to \mathsf{set}\, E \mid \\
&\qquad \forall x y \in \mathsf{set}\, D.\ \mathsf{rel}\, D\, x\, y \Rightarrow \mathsf{rel}\, E (f \diamond x)(f \diamond y)\} \\
&\mathsf{cont}(D, E) \stackrel{\text{def}}{=} \\
&\quad \{f \in \mathsf{mono}(D, E) \mid \\
&\qquad \forall X.\ \mathsf{chain}\, D\, X \Rightarrow f \diamond (\mathsf{lub}\, D\, X) = \mathsf{lub}\, E (\lambda n.\ f \diamond (X\, n))\}.
\end{aligned}$$

Note that monotonic and continuous functions are functions in set theory, not in HOL. Hence, the type of a continuous function is V (functions are sets). The second lub of the last definition makes sense since the result of applying a monotonic function to each element of a chain is itself a chain.

The continuous function space construction on cpos is defined as the pair consisting of the set of continuous functions between two cpos and the pointwise ordering relation on functions:

$$\mathsf{cf}(D, E) \stackrel{\text{def}}{=} \mathsf{cont}(D, E), \lambda f g. \forall x \in \mathsf{set}\, D.\ \mathsf{rel}\, E (f \diamond x)(g \diamond x).$$

The construction always yields a cpo if its arguments are cpos:

$$\forall DE.\ \mathsf{cpo}\, D \Rightarrow \mathsf{cpo}\, E \Rightarrow \mathsf{cpo}(\mathsf{cf}(D, E)).$$

The continuous function space is sometimes called the cpo of continuous functions, or just the function space.

Least upper bounds are calculated pointwise in the function space since the underlying relation is:

$$\begin{array}{l} \forall DEX. \\ \quad \mathsf{chain}(\mathsf{cf}(D, E))X \Rightarrow \mathsf{cpo}\, D \Rightarrow \mathsf{cpo}\, E \Rightarrow \\ \quad \mathsf{lub}(\mathsf{cf}(D, E))X = \lambda x \in \mathsf{set}\, D.\ \mathsf{lub}\, E(\lambda n.\, X\, n \diamond x). \end{array}$$

Besides, the cpo of continuous functions is pointed if the codomain is pointed

$$\forall DE.\ \mathsf{cpo}\, D \Rightarrow \mathsf{pcpo}\, E \Rightarrow \mathsf{pcpo}(\mathsf{cf}(D, E))$$

and its bottom element is the constant function which equals bottom everywhere.

3.3 Identity and composition

There are two standard constructions for writing continuous functions which are central to the present development. The identity function of set theory is a continuous function on any cpo:

$$\forall D.\ \mathsf{cpo}\, D \Rightarrow \mathsf{Id}(\mathsf{set}\, D) \in \mathsf{cont}(D, D).$$

And the composition function preserves continuity:

$$\begin{array}{l} \forall f g D D' E. \\ \quad f \in \mathsf{cont}(D', E) \Rightarrow g \in \mathsf{cont}(D, D') \Rightarrow \mathsf{cpo}\, D \Rightarrow \\ \quad f \mathbin{\mathsf{O}} g \in \mathsf{cont}(D, E). \end{array}$$

From these theorems we may conclude that cpos form a category with continuous functions as the arrows though this has no direct importance in this paper. (Note that in order to prove the second theorem it is enough to assume that the domain of the first function is a cpo.)

In fact, the composition function not only preserves continuity it is also itself a continuous function. It is monotonic, as stated by

$$\begin{array}{l} \forall f g f' g' D D' E. \\ \quad f \in \mathsf{cont}(D', E) \Rightarrow g \in \mathsf{cont}(D, D') \Rightarrow \\ \quad f' \in \mathsf{cont}(D', E) \Rightarrow g' \in \mathsf{cont}(D, D') \Rightarrow \mathsf{cpo}\, E \Rightarrow \\ \quad \mathsf{rel}(\mathsf{cf}(D', E))\, f\, f' \Rightarrow \mathsf{rel}(\mathsf{cf}(D, D'))\, g\, g' \Rightarrow \mathsf{rel}(\mathsf{cf}(D, E))(f \mathbin{\mathsf{O}} g)(f' \mathbin{\mathsf{O}} g'), \end{array}$$

and it preserves lubs of chains of continuous functions:

$$\forall X Y D D' E.$$
$$\mathsf{chain}(\mathsf{cf}(D', E))X \Rightarrow \mathsf{chain}(\mathsf{cf}(D, D'))Y \Rightarrow \mathsf{cpo}\, D \Rightarrow \mathsf{cpo}\, D' \Rightarrow \mathsf{cpo}\, E \Rightarrow$$
$$\mathsf{lub}(\mathsf{cf}(D', E))X \,\mathsf{O}\, \mathsf{lub}(\mathsf{cf}(D, D'))Y = \mathsf{lub}(\mathsf{cf}(D, E))(\lambda n.\, X\, n \,\mathsf{O}\, Y\, n).$$

These theorems were used several times in proofs. Note that it is not possible to state the continuity of O as membership of the set of continuous functions since O is a logical function (as noted in Section 2.3). It is the logical function which takes two set functions as arguments and returns a set function as a result.

3.4 Notes on alternative formalisations

The formalisation presented above is similar to the formalisation presented in [Age94b, Age93], yet slightly different. First of all, the type used to represent cpos is different. Earlier it was

$$(\alpha \to \mathit{bool})\#(\alpha \to \alpha \to \mathit{bool}),$$

now it is

$$V\#(V \to V \to \mathit{bool}).$$

Note an important difference here: in pure HOL the type representing sets of elements of some type is different from the type of the elements. This is not the case in the HOL-ST representation where elements of sets are themselves sets.

Another minor difference is that the present formalisation is less verbose. We have changed the notion of upper bound of some set to upper bound of some chain. Hence, we avoid taking the sets of elements of chains and the knowledge that elements of chains are always elements of a cpo also simplifies matters a bit. Besides, we do not introduce po and cpo conditions unless necessary. For instance, in the definition of continuous functions we do not require the domain and codomain to be cpos.

Some might feel that representing cpos as relations and defining the set of a cpo as the elements for which the relation is reflexive would be a simpler formalisation (cf. the discussion on alternative formalisations in [Age94b]). This approach is not possible when relations are HOL functions which take arguments of type V because none of the valid ways of forming sets seem to be applicable (a set must be constructed from other sets). Alternatively, we could have represented relations as relations of set theory (i.e. as sets of pairs). In this case, it would be possible and perhaps even natural to partly ignore the set component of cpos.

We did not attempt this, however, since in the formalisation presented in this paper we wanted to try to exploit higher order logic as much as possible. Hence, cpos are HOL pairs, ordering relations are HOL functions and chains are HOL functions from the HOL type of natural numbers *num*. Alternatively, we could have chosen to do more work in set theory. For instance, the natural number argument of chains could have been in the set Num and cpos could have been represented by relations of set theory as indicated above.

It is not essential whether one uses a relation of set theory or a certain pair of higher order logic to represent cpos. In both cases, I think that the constants rel and set should be used to support the view that a cpo is a set with an associated relation. However, it *does* make a difference whether one uses natural number of set theory or higher order logic, and in general, whether one uses sets of HOL-ST or types of pure HOL. Using the latter type assumptions are avoided and furthermore, type checking is done automatic in ML. Using sets in HOL-ST, type checking, i.e. checking terms are in the right sets, is performed (late) by theorem proving.

Obviously, if we wish to exploit the additional power of set theory we sometimes have to leave higher order logic and pay the price. It is an interesting and difficult question which parts of the formalisation should be done in set theory and which should be done in higher order logic. The question is discussed further in the following section.

4 The inverse limit construction

Just as chains of elements of cpos have least upper bounds we can consider "chains" of cpos which have "least upper bounds", called *inverse limits*. The ordering relation on elements is replaced by the notion of embedding morphisms (determined in embedding projection pairs). A certain constant Dinf parametrised by a chain of cpos can be proven once and forall to yield the inverse limit. Dinf is defined as a subcpo of an infinite cartesian product of cpos. Hence, the elements of Dinf are infinite tuples with components which may be in different cpos. In this section, we present the definitions and theorems which formalise the inverse limit construction in HOL-ST.

4.1 A cpo of infinite tuples

The infinite cartesian product of an infinite sequence of cpos is defined using the dependent product construction on sets. Before the infinite product construction can be defined we must make a decision: Should sequences of cpos, written as $\mathbf{D}$, be formalised as functions from the *type* of numbers or from the *set* of numbers. The latter choice seems most obvious since the elements of the dependent product

$$\prod_{x \in X} Y\, x$$

are functions of set theory mapping elements $x \in X$ to elements $y \in Y\, x$. Hence, the infinite product could be defined as follows:

$$\prod_{n \in \mathsf{Num}} \mathsf{set}(\mathbf{D}\, n).$$

The alternative would be

$$\prod_{n \in \mathsf{Num}} \mathsf{set}(\mathbf{D}(\mathsf{Num2num}\, n))$$

which seems awkward at first because though the sequence takes arguments in *num* we must still use Num in the dependent product. And indeed, the first of the many theorems we wish to prove about the formalisation of infinite products and inverse limits become more complicated when we choose this second approach. However, in the long run it is worth paying the price of inconvenience in the beginning since things become quite horrible later on if we choose the first approach. In fact, we did both developments to test out which approach was the simpler. Since the latter was, this paper will only present it.

There are perhaps three main reasons why the *num* approach was simpler. First, we avoid explicit type assumptions of the form $n \in \mathsf{Num}$ and furthermore, we avoid type checking such assumptions by theorem proving. Second, we avoid redeveloping parts of the HOL theory of natural numbers in set theory (addition, ordering relations, recursive function definitions) though this can be done in a straightforward way, I guess, using the isomorphisms (see Section 5 of [Gor94]). A third reason is that in the previous section we chose to represent ordinary sequences (chains) of elements of cpos as functions from the type of numbers *num*. This was natural since there is no need to use Num here, and in general, using HOL types is simpler as discussed above. If we chose to use the set Num for sequences of cpos this would give an unfortunate mismatch. We would have to do a lot of translation between the two kinds of numbers and for convenience we would have to define alternative versions of chain and lub to work on Num. For these reasons, chains of cpos are represented as functions from HOL numbers.

Now, we should be ready to define the infinite cartesian product construction on cpos which associates the pointwise ordering with elements of the dependent product:

$$\begin{aligned}&\mathsf{iprod}\,\mathbf{D} \stackrel{\mathrm{def}}{=}\\ &\quad \prod_{n \in \mathsf{Num}} \mathsf{set}(\mathbf{D}(\mathsf{Num2num}\,n)),\\ &\quad \lambda xy.\,\forall n.\ \mathsf{rel}(\mathbf{D}\,n)(x \diamond (\mathsf{num2Num}\,n))(y \diamond (\mathsf{num2Num}\,n)).\end{aligned}$$

Note that all number conversions could be eliminated if we used Num rather than *num* to represent sequences. This illustrates how difficult a decision it was to choose the *num* approach. Elements of the infinite product can be viewed as infinite tuples of elements since they are functions from natural numbers to some cpo (dependent on the number argument). Indeed, iprod does yield a construction on cpos and on pointed cpos as well:

$$\begin{aligned}&\forall \mathbf{D}.\ (\forall n.\ \mathsf{cpo}(\mathbf{D}\,n)) \Rightarrow \mathsf{cpo}(\mathsf{iprod}\,\mathbf{D})\\ &\forall \mathbf{D}.\ (\forall n.\ \mathsf{pcpo}(\mathbf{D}\,n)) \Rightarrow \mathsf{pcpo}(\mathsf{iprod}\,\mathbf{D}).\end{aligned}$$

Actually, the fact that we have been able to define iprod is interesting in itself. This is not possible in pure HOL for instance since dependent types are not available. Finally, note that if we had represented sequences as functions from Num then the quantifications '$\forall n$' above would have to be restricted to Num (otherwise they would be quantifying over all sets).

4.2 Embedding projection pairs

The ordering on elements of a cpo is generalised to the notion of embedding morphism on cpos. Embeddings come in pairs with projections, forming the so-called embedding projection pairs:

$$\begin{aligned}&\mathsf{projpair}(D,E)(e,p) \stackrel{\mathrm{def}}{=}\\ &\quad e \in \mathsf{cont}(D,E) \wedge p \in \mathsf{cont}(E,D) \wedge\\ &\quad p \mathbin{\mathsf{O}} e = \mathsf{Id}(\mathsf{set}\,D) \wedge \mathsf{rel}(\mathsf{cf}(E,E))(e \mathbin{\mathsf{O}} p)(\mathsf{Id}(\mathsf{set}\,E)).\end{aligned}$$

An embedding projection pair (e,p) between cpos D and E may be picturised as follows:

$$D \underset{p}{\overset{e}{\rightleftarrows}} E.$$

The conditions make sure that the structure of E is richer than that of D (and can contain it). An embedding is one-one and a projection is onto. Furthermore, if the cpos are pointed then both functions are strict.

Embeddings uniquely determine projections and vice versa:

$$
\begin{array}{l}
\forall D E e p e' p'. \\
\quad \mathsf{cpo}\, D \Rightarrow \mathsf{cpo}\, E \Rightarrow \mathsf{projpair}(D,E)(e,p) \Rightarrow \mathsf{projpair}(D,E)(e',p') \Rightarrow \\
\quad (e = e') = (p = p').
\end{array}
$$

Hence, it is enough to consider embeddings

$$\mathsf{emb}(D,E)e \stackrel{\text{def}}{=} \exists p.\ \mathsf{projpair}(D,E)(e,p)$$

and define the associated projections, or *retracts* as they are often called, using the choice operator:

$$\mathsf{R}(D,E)e \stackrel{\text{def}}{=} \varepsilon p.\ \mathsf{projpair}(D,E)(e,p).$$

When we wish to be less formal, we will sometimes write $\mathsf{emb}(D,E)e$ using the standard mathematical notation $e : D \lhd E$. Similarly, $\mathsf{R}(D,E)e$ is sometimes written as e^R.

The identity function is a simple example of an embedding:

$$
\begin{array}{l}
\forall D.\ \mathsf{cpo}\, D \Rightarrow \mathsf{emb}(D,D)(\mathsf{Id}(\mathsf{set}\, D)) \\
\forall D.\ \mathsf{cpo}\, D \Rightarrow \mathsf{R}(D,D)(\mathsf{Id}(\mathsf{set}\, D)) = \mathsf{Id}(\mathsf{set}\, D).
\end{array}
$$

Obviously, the associated projection is the identity function itself.

In order to prove a continuous function is an embedding we sometimes have to display the associated projection explicitly, but not always. For instance, once we have proved that composition preserves embeddings

$$
\begin{array}{l}
\forall e e' D D' E. \\
\quad \mathsf{emb}(D,D')e \Rightarrow \mathsf{emb}(D',E)e' \Rightarrow \mathsf{cpo}\, D \Rightarrow \mathsf{cpo}\, D' \Rightarrow \mathsf{cpo}\, E \Rightarrow \\
\quad \mathsf{emb}(D,E)(e' \mathsf{O}\, e)
\end{array}
$$

we can derive new embeddings directly without displaying the projection. However, it is often useful to be able to calculate what the associated projections are:

$$
\begin{array}{l}
\forall e e' D D' E. \\
\quad \mathsf{emb}(D,D')e \Rightarrow \mathsf{emb}(D',E)e' \Rightarrow \mathsf{cpo}\, D \Rightarrow \mathsf{cpo}\, D' \Rightarrow \mathsf{cpo}\, E \Rightarrow \\
\quad \mathsf{R}(D,E)(e' \mathsf{O}\, e) = \mathsf{R}(D,D')e \,\mathsf{O}\, \mathsf{R}(D',E)e'.
\end{array}
$$

Hence, the projections of compositions of embeddings are obtained by composing the projections of each embedding in the reversed order.

4.3 Embedding projection chains of cpos

Embeddings are used to form chains of cpos in a similar way that the ordering on elements of cpos is used to form chains. A chain of cpos is a pair $(\mathbf{D}, \mathbf{e})$ consisting of a sequence of cpos D_n and a sequence of embeddings e_n where $e_n : D_n \lhd D_{n+1}$ for all $n : num$:

$$D_0 \overset{e_0}{\lhd} D_1 \overset{e_1}{\lhd} \cdots \overset{e_{n-1}}{\lhd} D_n \overset{e_n}{\lhd} \cdots.$$

For convenience, we use $\mathbf{D}$ and $\mathbf{e}$ for sequences and, perhaps confusingly, D_n and e_n for each of their respective elements. The notion of embedding projection chain of cpos is formalised as follows in HOL-ST:

$$\mathsf{emb\_chain}\,\mathbf{D}\,\mathbf{e} \overset{\mathrm{def}}{=} (\forall n.\ \mathsf{cpo}(\mathbf{D}\,n)) \wedge (\forall n.\ \mathsf{emb}(\mathbf{D}\,n, \mathbf{D}(\mathsf{SUC}\,n))(\mathbf{e}\,n)).$$

We shall introduce a generalisation of the embeddings associated with chains of cpos since these just convert between two consecutive cpos of a chain. Exploiting that composition preserves embeddings, we can define a function eps to convert between any two cpos D_m and D_n:

$$\mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,m\,n = \begin{cases} e_{n-1} \mathsf{O} \cdots \mathsf{O}\, e_m & \text{if } m < n \\ \mathsf{Id}(\mathsf{set}\,D_m) & \text{if } m = n \\ e_n^R \,\mathsf{O} \cdots \mathsf{O}\, e_{m-1}^R & \text{if } m > n. \end{cases}$$

If $m \leq n$ then eps gives an embedding:

$$\begin{array}{l} \forall \mathbf{D}\mathbf{e}. \\ \quad \mathsf{emb\_chain}\,\mathbf{D}\,\mathbf{e} \Rightarrow \\ \quad \forall mn.\ m \leq n \Rightarrow \mathsf{emb}(\mathbf{D}\,m, \mathbf{D}\,n)(\mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,m\,n). \end{array}$$

If we use Num instead of num then this theorem would be stated as follows:

$$\begin{array}{l} \forall \mathbf{D}\mathbf{e}. \\ \quad (\forall n.\ n \in \mathsf{Num} \Rightarrow \mathsf{cpo}(\mathbf{D}\,n)) \Rightarrow \\ \quad (\forall n.\ n \in \mathsf{Num} \Rightarrow \mathsf{emb}(\mathbf{D}\,n, \mathbf{D}(\mathsf{Suc}\,n))(\mathbf{e}\,n)) \Rightarrow \\ \quad \forall mn.\ m \in \mathsf{Num} \Rightarrow n \in \mathsf{Num} \Rightarrow m \downarrow\!\leq n \Rightarrow \\ \qquad \mathsf{emb}(\mathbf{D}\,m, \mathbf{D}\,n)(\mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,m\,n) \end{array}$$

where $\downarrow$ converts a relation to set theory. Note the additional type assumptions (the definition of $\mathsf{emb\_chain}$ is expanded for illustration). Much effort was put into defining and reasoning about eps. In particular, the following two important theorems had quite long and messy proofs:

$$\begin{aligned}
&\forall \mathbf{D}\mathbf{e}.\\
&\quad \mathsf{emb_chain}\,\mathbf{D}\,\mathbf{e} \Rightarrow\\
&\quad \forall mnk.\ m \leq k \Rightarrow \mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,m\,n = \mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,k\,n \,\mathsf{O}\, \mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,m\,k\\
&\forall \mathbf{D}\mathbf{e}.\\
&\quad \mathsf{emb_chain}\,\mathbf{D}\,\mathbf{e} \Rightarrow\\
&\quad \forall mnk.\ n \leq k \Rightarrow \mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,m\,n = \mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,k\,n \,\mathsf{O}\, \mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,m\,k.
\end{aligned}$$

The theorems state how the function $\mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,m\,n$ mapping elements of D_m to element of D_n can be split to go via D_k for certain k. We shall not go into the boring details of proving properties of eps here but just mention that it was defined by two primitive recursive definitions on *num*. Using Num to represent sequences complicated reasoning about eps a lot since its definition and properties rely heavily on natural numbers. For instance, at the moment there is no way to define primitive recursive functions or do induction in set theory so a lot of translation from *num* to Num had to be done. On the other hand, using induction in set theory one would have the inconvenience of proof obligations due to typing assumptions.

4.4 Dinf—the inverse limit

Given a chain of cpos $(\mathbf{D}, \mathbf{e})$, the inverse limit is defined as a subcpo of the infinite cartesian product of $\mathbf{D}$. First, we introduce the notions of subcpo and subpcpo ('p' for pointed) briefly:

$$\begin{aligned}
&\mathsf{subcpo}\,D\,E \stackrel{\text{def}}{=}\\
&\quad \mathsf{set}\,D \subseteq \mathsf{set}\,E \wedge \mathsf{rel}\,D = \mathsf{rel}\,E \wedge \forall X.\ \mathsf{chain}\,D\,X \Rightarrow \mathsf{lub}\,E\,X \in \mathsf{set}\,D\\
&\mathsf{subpcpo}\,D\,E \stackrel{\text{def}}{=} \mathsf{subcpo}\,D\,E \wedge \mathsf{bot}\,E \in \mathsf{set}\,D.
\end{aligned}$$

A constructor can be introduced for defining both subcpos and subpcpos

$$\mathsf{mkcpo}\,D\,P \stackrel{\text{def}}{=} \{x \in \mathsf{set}\,D \mid P\,x\}, \mathsf{rel}\,D$$

as stated by the following two theorems:

$$\begin{aligned}
&\forall DP.\\
&\quad (\forall X.\ \mathsf{chain}(\mathsf{mkcpo}\,D\,P)X \Rightarrow P(\mathsf{lub}\,D\,X)) \Rightarrow \mathsf{cpo}\,D \Rightarrow\\
&\quad \mathsf{subcpo}(\mathsf{mkcpo}\,D\,P)D
\end{aligned}$$

$$
\begin{aligned}
&\forall D P.\\
&\quad (\forall X.\ \mathsf{chain}(\mathsf{mkcpo}\,D\,P)X \Rightarrow P(\mathsf{lub}\,D\,X)) \Rightarrow P(\mathsf{bot}\,D) \Rightarrow \mathsf{pcpo}\,D \Rightarrow\\
&\quad \mathsf{subpcpo}(\mathsf{mkcpo}\,D\,P)D.
\end{aligned}
$$

Hence, using mkcpo it is enough to prove that the specified subset of a (pointed) cpo contains lubs (and bottom).

By defining Dinf using mkcpo we are therefore able to obtain quite easily that it yields cpos and pointed cpos because iprod does. The definition is:

$$
\begin{aligned}
&\mathsf{Dinf}\,\mathbf{D}\,\mathbf{e} \stackrel{\mathrm{def}}{=}\\
&\quad \mathsf{mkcpo}\\
&\qquad (\mathsf{iprod}\,\mathbf{D})\\
&\qquad (\lambda x.\forall n.\\
&\qquad\quad \mathsf{R}(\mathbf{D}\,n, \mathbf{D}(\mathsf{SUC}\,n))(\mathbf{e}\,n) \diamond (x \diamond (\mathsf{num2Num}(\mathsf{SUC}\,n))) =\\
&\qquad\quad x \diamond (\mathsf{num2Num}\,n)),
\end{aligned}
$$

where the annoying num2Num conversions could be avoided if we had chosen to use Num instead of *num*. Informally, the underlying set of Dinf is defined as the subset of all infinite tuples x on which the n-th projection e_n^R maps the $(n+1)$-st index to the n-th index for all n: $e_n^R(x_{n+1}) = x_n$. The fact that Dinf always yields a cpo is stated as follows:

$$\forall \mathbf{D}\mathbf{e}.\ \mathsf{emb\_chain}\,\mathbf{D}\,\mathbf{e} \Rightarrow \mathsf{cpo}(\mathsf{Dinf}\,\mathbf{D}\,\mathbf{e}).$$

We can replace cpo by pcpo in this theorem to obtain the same result for pointed cpos.

Given a chain of cpos, we wish to prove that the pair consisting of Dinf and a certain family of embeddings from D_n to Dinf is an inverse limit, i.e. this pair satisfies a certain commuting diagram condition and is universal with this property. The desired family of embeddings is defined using the eps mappings as follows:

$$\mathsf{rho}\,\mathbf{D}\,\mathbf{e}\,n \stackrel{\mathrm{def}}{=} \lambda x \in \mathsf{set}(\mathbf{D}\,n).\,\lambda m \in \mathsf{Num}.\,\mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,n(\mathsf{Num2num}\,m) \diamond x$$

So, given an element $x \in D_n$ the function $\mathsf{rho}\,\mathbf{D}\,\mathbf{e}\,n$ returns a tuple of the following form:

$$(\ldots,\ e_{n-2}^R(e_{n-1}^R(x)),\ e_{n-1}^R(x),\ x,\ e_n(x),\ e_{n+1}(e_n(x)),\ \ldots)$$

where the n-th element is x. The projection associated with $\mathsf{rho}\,\mathbf{D}\,\mathbf{e}\,n$ simply takes the n-th component of the tuple so obviously we have

$$\mathsf{rho}^R_{(\mathbf{D},\mathbf{e})} n \ \mathsf{O}\ \mathsf{rho}_{(\mathbf{D},\mathbf{e})} n = \mathsf{Id}(\mathsf{set} D_n)$$

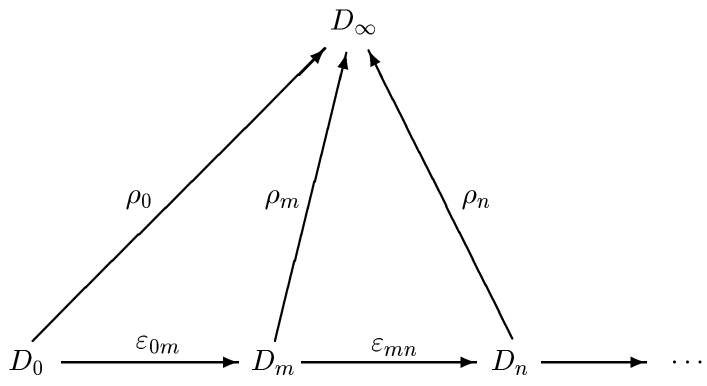

Figure 1: The commutivity of the inverse limit D_∞ with embeddings ρ_n.

where we have subscripted $\mathbf{D}$ and $\mathbf{e}$ for convenience (we shall continue to do this a few times below). Furthermore, we can prove that $\mathsf{rho}\,\mathbf{D}\,\mathbf{e}\,n$ indeed is an embedding from D_n to Dinf:

$$\forall \mathbf{D}\mathbf{e}.\ \mathsf{emb\_chain}\,\mathbf{D}\,\mathbf{e} \Rightarrow \forall n.\ \mathsf{emb}(\mathbf{D}\,n, \mathsf{Dinf}\,\mathbf{D}\,\mathbf{e})(\mathsf{rho}\,\mathbf{D}\,\mathbf{e}\,n)$$

Hence, we have defined a family of embeddings from D_n to Dinf. Next, we must prove Dinf (with this family) enjoys the desired properties: commutivity and universality.

4.5 Commutivity

The commuting diagram property that we wish to prove about Dinf and rho can be picturised as in Figure 1 and written as

$$\forall \mathbf{D}\mathbf{e}.\ \mathsf{emb\_chain}\,\mathbf{D}\,\mathbf{e} \Rightarrow \mathsf{commute}\,\mathbf{D}\,\mathbf{e}(\mathsf{Dinf}\,\mathbf{D}\,\mathbf{e})(\mathsf{rho}\,\mathbf{D}\,\mathbf{e})$$

where $\mathsf{commute}$ is defined by

$$\begin{array}{l}\mathsf{commute}\,\mathbf{D}\,\mathbf{e}\,E\,\mathbf{r} \stackrel{\mathrm{def}}{=} \\ \qquad (\forall n.\ \mathsf{emb}(\mathbf{D}\,n, E)(\mathbf{r}\,n)) \wedge (\forall mn.\ m \leq n \Rightarrow \mathbf{r}\,n\,\mathsf{O}\,\mathsf{eps}\,\mathbf{D}\,\mathbf{e}\,m\,n = \mathbf{r}\,m).\end{array}$$

The main step of the proof is the use of the fact that eps can be split (see above).

$$
\begin{tikzcd}
 & & G & & \\
 & & D_\infty \arrow[u, dashed, "\exists!\theta"'] & & \\
\cdots \arrow[r] & D_m \arrow[rr, "\varepsilon_{mn}"] \arrow[uur, "\phi_m"] \arrow[ur, "\rho_m"'] & & D_n \arrow[r] \arrow[uul, "\phi_n"'] \arrow[ul, "\rho_n"] & \cdots
\end{tikzcd}
$$

Figure 2: Universality of D_∞ with embeddings ρ_n.

4.6 Universality

The more difficult part of proving that Dinf (with rho) is the inverse limit is proving that it is universal. This property can be picturised as in Figure 2 which informally states that for any cpo G which satisfy the commuting diagram condition with a family of embeddings ϕ, there is a unique embedding $\theta : \mathsf{Dinf}_{(\mathbf{D},\mathbf{e})} \to G$ such that θ factors ϕ_n: $\phi_n = \theta \mathbin{\mathsf{O}} \mathsf{rho}_{(\mathbf{D},\mathbf{e})} n$ for all $n : num$. An embedding with this property is called a mediating morphism:

$$\mathsf{mediating}(E, G, \mathbf{r}, \mathbf{f})t \stackrel{\text{def}}{=} \mathsf{emb}(E, G)t \wedge \forall n.\ \mathbf{f}\, n = t \mathbin{\mathsf{O}} \mathbf{r}\, n.$$

The mediating morphism which ensures Dinf is universal appears in the following theorem:

$$
\begin{array}{l}
\forall \mathbf{D}\mathbf{e}G\mathbf{f}. \\
\quad \mathsf{emb\_chain}\,\mathbf{D}\,\mathbf{e} \Rightarrow \mathsf{cpo}\,G \Rightarrow \mathsf{commute}\,\mathbf{D}\,\mathbf{e}\,G\,\mathbf{f} \Rightarrow \\
\quad \mathsf{mediating} \\
\qquad (\mathsf{Dinf}\,\mathbf{D}\,\mathbf{e}, G, \mathsf{rho}\,\mathbf{D}\,\mathbf{e}, \mathbf{f}) \\
\qquad (\mathsf{lub}(\mathsf{cf}(\mathsf{Dinf}\,\mathbf{D}\,\mathbf{e}, G))(\lambda n.\, \mathbf{f}\, n \mathbin{\mathsf{O}} \mathsf{R}(\mathbf{D}\, n, \mathsf{Dinf}\,\mathbf{D}\,\mathbf{e})(\mathsf{rho}\,\mathbf{D}\,\mathbf{e}\, n))).
\end{array}
$$

The uniqueness has also been proved though it is not stated here. It takes some amount of effort to prove the desired properties of this mediating morphism which is the lub of a sequence of functions. First of all, the sequence must be proven to form a chain of continuous functions, otherwise the result of applying lub is not necessarily a least upper bound. Next, the associated projection must be constructed and the pair must be proved to be an embedding projection pair. Hence, the proofs involve a lot of reasoning about lubs and composition.

A main part of proving the universality of Dinf with rho is realising that the functions $\mathsf{rho}\, n \,\mathsf{O}\, \mathsf{rho}^R\, n$ form a chain which converges to the identity function on Dinf:

$$
\begin{array}{l}
\forall \mathbf{D}\mathbf{e}. \\
\quad \mathsf{emb\_chain}\, \mathbf{D}\, \mathbf{e} \Rightarrow \\
\quad \mathsf{lub}(\mathsf{cf}(\mathsf{Dinf}\, \mathbf{D}\, \mathbf{e}, \mathsf{Dinf}\, \mathbf{D}\, \mathbf{e})) \\
\qquad (\lambda n.\, \mathsf{rho}\, \mathbf{D}\, \mathbf{e}\, n \,\mathsf{O}\, \mathsf{R}(\mathbf{D}\, n, \mathsf{Dinf}\, \mathbf{D}\, \mathbf{e})(\mathsf{rho}\, \mathbf{D}\, \mathbf{e}\, n)) = \mathsf{Id}(\mathsf{set}(\mathsf{Dinf}\, \mathbf{D}\, \mathbf{e})).
\end{array}
$$

Then the universality follows from the fact that this condition implies universality in general, i.e. any commuting diagram with this property is universal:

$$
\begin{array}{l}
\forall \mathbf{D}\mathbf{e}E\mathbf{r}G\mathbf{f}. \\
\quad \mathsf{emb\_chain}\, \mathbf{D}\, \mathbf{e} \Rightarrow \\
\quad \mathsf{cpo}\, E \Rightarrow \mathsf{commute}\, \mathbf{D}\, \mathbf{e}\, E\, \mathbf{r} \Rightarrow \\
\quad \mathsf{lub}(\mathsf{cf}(E, E))(\lambda n.\, \mathbf{r}\, n \,\mathsf{O}\, \mathsf{R}(\mathbf{D}\, n, E)(\mathbf{r}\, n)) = \mathsf{Id}(\mathsf{set}\, E) \Rightarrow \\
\quad \mathsf{cpo}\, G \Rightarrow \mathsf{commute}\, \mathbf{D}\, \mathbf{e}\, G\, \mathbf{f} \Rightarrow \\
\quad \mathsf{mediating}(E, G, \mathbf{r}, \mathbf{f})(\mathsf{lub}(\mathsf{cf}(E, G))(\lambda n.\, \mathbf{f}\, n \,\mathsf{O}\, \mathsf{R}(\mathbf{D}\, n, E)(\mathbf{r}\, n))) \wedge \\
\quad \forall t.\, \mathsf{mediating}(E, G, \mathbf{r}, \mathbf{f})t \Rightarrow \\
\qquad t = \mathsf{lub}(\mathsf{cf}(E, G))(\lambda n.\, \mathbf{f}\, n \,\mathsf{O}\, \mathsf{R}(\mathbf{D}\, n, E)(\mathbf{r}\, n)).
\end{array}
$$

This theorem is called the *lub-implies-universal* theorem below. The implication can be strengthened to equality but it has not been necessary to do this in the present development.

We have now completed the formalisation of the inverse limit construction in HOL-ST. A cpo $\mathsf{Dinf}_{(\mathbf{D},\mathbf{e})}$ has been defined and proved to be the inverse limit of an arbitrary chain of cpos $(\mathbf{D}, \mathbf{e})$. Our next step is to show how this result can be used to solve recursive domain equations in HOL-ST. We will focus on one example, namely $D \cong [D \to D]$, but the inverse limit construction can be exploited for any equation defining a functor on the category of cpos which preserves embeddings and the universal property.

5 Continuity of the continuous functions space

In general, a recursive domain (isomorphism) equation can have the form

$$D \cong \mathcal{F}(D)$$

where $\mathcal{F}$ is a continuous covariant functor. This means that $\mathcal{F}$ is a pair consisting of a construction on cpos and a construction on embeddings which satisfies certain conditions. As a constructor on embeddings, it must preserve identity and composition to be a *covariant functor* (a term of category theory). And to be a *continuous* covariant functor it must preserve universality which implies that it preserves inverse limits of chains of cpos.

We should remark that a domain equation is stated using the cpo construction part of a functor, for instance, using the continuous function space construction as in

$$D \cong [D \to D],$$

and the associated construction on embeddings is only implicit here. Yet, the construction on embeddings is very important. If D is a cpo which satisfies the commutativity and universality properties with some family of embeddings then the construction is used to construct a family of embeddings which makes the domain $\mathcal{F}(D)$ satisfy the same two properties. As a consequence, the inverse limit, formalised as Dinf in HOL-ST, yields a kind of 'semi-solution' of the isomorphism equation. More precisely, the functor $\mathcal{F}$ preserves inverse limits. For instance, to continue the example above, the fact that the function space preserves inverse limits is stated roughly as follows in the HOL-ST formalisation:

$$\mathsf{Dinf}_{(\ldots)} \cong \mathsf{cf}(\mathsf{Dinf}_{(\mathbf{D},\mathbf{e})}, \mathsf{Dinf}_{(\mathbf{D},\mathbf{e})}),$$

where the notion of isomorphism between domains is defined as usual:

$$\begin{aligned} D \cong E \stackrel{\text{def}}{=} \exists f g.\ & f \in \mathsf{cont}(D, E) \land g \in \mathsf{cont}(E, D) \land \\ & g \mathbin{\mathsf{O}} f = \mathsf{Id}(\mathsf{set}\, D) \land f \mathbin{\mathsf{O}} g = \mathsf{Id}(\mathsf{set}\, E). \end{aligned}$$

The arguments of the first Dinf, written as dots above, is not the chain $(\mathbf{D}, \mathbf{e})$ and therefore Dinf does not yield a real solution directly. However, a solution is obtained fairly easily from this fact. The main result of this section is a slightly more general version of this theorem which states precisely that the functor for the continuous function space preserves inverse limits of chains of cpos.

5.1 Covariance

The functor for the continuous function space consists of a construction on cpos and a construction on embeddings. We already defined the construction on cpos, called cf. The other part of the functor, the construction on embeddings, is defined as follows:

$$\mathsf{cf\_emb}(D, D', E)(e, e') \stackrel{\text{def}}{=} \lambda f \in \mathsf{cont}(D, D').\, e' \mathsf{O} f \mathsf{O} \mathsf{R}(D, E)e.$$

The cpo construction takes two cpos as arguments and similarly the construction on embeddings takes two embeddings as arguments. The idea is that if e is an embedding of D into E and e' is an embedding of D' into E' then $\mathsf{cf\_emb}_{(D,D',E)}(e, e')$ is an embedding of the continuous function space $\mathsf{cf}(D, D')$ into the continuous function space $\mathsf{cf}(E, E')$:

$$\begin{array}{l}\forall e e' D E D' E'.\\ \quad \mathsf{emb}(D, E)e \Rightarrow \mathsf{cpo}\, D \Rightarrow \mathsf{cpo}\, E \Rightarrow\\ \quad \mathsf{emb}(D', E')e' \Rightarrow \mathsf{cpo}\, D' \Rightarrow \mathsf{cpo}\, E' \Rightarrow\\ \quad \mathsf{emb}(\mathsf{cf}(D, D'), \mathsf{cf}(E, E'))(\mathsf{cf\_emb}(D, D', E)(e, e')).\end{array}$$

Hence, $\mathsf{cf\_emb}$ preserves embeddings. Furthermore, as a function on embeddings it preserves the identity function in each argument

$$\begin{array}{l}\forall D E.\\ \quad \mathsf{cpo}\, D \Rightarrow \mathsf{cpo}\, E \Rightarrow\\ \quad \mathsf{cf\_emb}(D, E, D)(\mathsf{Id}(\mathsf{set}\, D), \mathsf{Id}(\mathsf{set}\, E)) = \mathsf{Id}(\mathsf{cont}(D, E))\end{array}$$

and the composition of two embeddings, which is itself an embedding, is also preserved in each argument:

$$\begin{array}{l}\forall e_1 e_2 e_1' e_2' D D_1 E D' D_1' E'.\\ \quad \mathsf{emb}(D, D_1)e_1 \Rightarrow \mathsf{emb}(D_1, E)e_2 \Rightarrow \mathsf{cpo}\, D \Rightarrow \mathsf{cpo}\, D_1 \Rightarrow \mathsf{cpo}\, E \Rightarrow\\ \quad \mathsf{emb}(D', D_1')e_1' \Rightarrow \mathsf{emb}(D_1', E')e_2' \Rightarrow \mathsf{cpo}\, D' \Rightarrow \mathsf{cpo}\, D_1' \Rightarrow \mathsf{cpo}\, E' \Rightarrow\\ \quad \mathsf{cf\_emb}(D, D', E)(e_2 \mathsf{O} e_1, e_2' \mathsf{O} e_1') =\\ \quad \mathsf{cf\_emb}(D_1, D_1', E)(e_2, e_2') \mathsf{O} \mathsf{cf\_emb}(D, D', D_1)(e_1, e_1').\end{array}$$

This justifies that $\mathsf{cf\_emb}$ is a covariant functor. If we were working with a construction of only one argument then the theorems would be less complex and only have half the assumptions.

5.2 Continuity

As a first step towards proving continuity, i.e. commutivity and universality, of the functor for the function space we must derive that given two embedding chains of cpos $(\mathbf{D}, \mathbf{e})$ and $(\mathbf{D}', \mathbf{e}')$ the continuous function space construction can be used to construct another chain of cpos

$$\lambda n.\, \mathsf{cf}(D_n, D'_n),\ \lambda n.\, \mathsf{cf\_emb}_{(D_n, D'_n, D_{n+1})}(e_n, e'_n)$$

as stated by the theorem:

$$\begin{array}{l}
\forall \mathbf{e}\mathbf{e}'\mathbf{D}\mathbf{D}'. \\
\quad \mathsf{emb_chain}\, \mathbf{D}\, \mathbf{e} \Rightarrow \mathsf{emb_chain}\, \mathbf{D}'\, \mathbf{e}' \Rightarrow \\
\quad \mathsf{emb_chain}(\lambda n.\, \mathsf{cf}(\mathbf{D}\, n, \mathbf{D}'\, n)) \\
\qquad\qquad (\lambda n.\, \mathsf{cf_emb}(\mathbf{D}\, n, \mathbf{D}'\, n, \mathbf{D}(\mathsf{SUC}\, n))(\mathbf{e}\, n, \mathbf{e}'\, n)).
\end{array}$$

Assuming next that there are cpos E and E' with families of embeddings $r_n : D_n \lhd E$ and $r'_n : D'_n \lhd E'$ which satisfy the commuting diagram condition then these families and $\mathsf{cf\_emb}$ can be used to construct a family of embeddings such that $\mathsf{cf}(E, E')$ satisfies commutivity with this family:

$$\begin{array}{l}
\forall \mathbf{e}\mathbf{e}'\mathbf{r}\mathbf{r}'\mathbf{D}\mathbf{D}'EE'. \\
\quad \mathsf{emb_chain}\, \mathbf{D}\, \mathbf{e} \Rightarrow \mathsf{cpo}\, E \Rightarrow \mathsf{commute}\, \mathbf{D}\, \mathbf{e}\, E\, \mathbf{r} \Rightarrow \\
\quad \mathsf{emb_chain}\, \mathbf{D}'\, \mathbf{e}' \Rightarrow \mathsf{cpo}\, E' \Rightarrow \mathsf{commute}\, \mathbf{D}'\, \mathbf{e}'\, E'\, \mathbf{r}' \Rightarrow \\
\quad \mathsf{commute} \\
\qquad (\lambda n.\, \mathsf{cf}(\mathbf{D}\, n, \mathbf{D}'\, n)) \\
\qquad (\lambda n.\, \mathsf{cf_emb}(\mathbf{D}\, n, \mathbf{D}'\, n, \mathbf{D}(\mathsf{SUC}\, n))(\mathbf{e}\, n, \mathbf{e}'\, n)) \\
\qquad (\mathsf{cf}(E, E')) \\
\qquad (\lambda n.\, \mathsf{cf_emb}(\mathbf{D}\, n, \mathbf{D}'\, n, E)(\mathbf{r}\, n, \mathbf{r}'\, n)).
\end{array}$$

Furthermore, $\mathsf{cf}(E, E')$ is universal. This is obtained from the following theorem

$$\begin{array}{l}
\forall \mathbf{e}\mathbf{e}'\mathbf{r}\mathbf{r}'\mathbf{D}\mathbf{D}'EE'. \\
\quad \mathsf{emb_chain}\, \mathbf{D}\, \mathbf{e} \Rightarrow \mathsf{cpo}\, E \Rightarrow \mathsf{commute}\, \mathbf{D}\, \mathbf{e}\, E\, \mathbf{r} \Rightarrow \\
\quad \mathsf{lub}(\mathsf{cf}(E, E))(\lambda n.\, \mathbf{r}\, n\, \mathsf{O}\, \mathsf{R}(\mathbf{D}\, n, E)(\mathbf{r}\, n)) = \mathsf{Id}(\mathsf{set}\, E) \Rightarrow \\
\quad \mathsf{emb_chain}\, \mathbf{D}'\, \mathbf{e}' \Rightarrow \mathsf{cpo}\, E' \Rightarrow \mathsf{commute}\, \mathbf{D}'\, \mathbf{e}'\, E'\, \mathbf{r}' \Rightarrow \\
\quad \mathsf{lub}(\mathsf{cf}(E', E'))(\lambda n.\, \mathbf{r}'\, n\, \mathsf{O}\, \mathsf{R}(\mathbf{D}'\, n, E')(\mathbf{r}'\, n)) = \mathsf{Id}(\mathsf{set}\, E') \Rightarrow \\
\quad \mathsf{lub}(\mathsf{cf}(\mathsf{cf}(E, E'), \mathsf{cf}(E, E'))) \\
\qquad (\lambda n.\, \mathsf{cf_emb}(\mathbf{D}\, n, \mathbf{D}'\, n, E)(\mathbf{r}\, n, \mathbf{r}'\, n)\, \mathsf{O} \\
\qquad\qquad \mathsf{R}(\mathsf{cf}(\mathbf{D}\, n, \mathbf{D}'\, n), \mathsf{cf}(E, E'))(\mathsf{cf_emb}(\mathbf{D}\, n, \mathbf{D}'\, n, E)(\mathbf{r}\, n, \mathbf{r}'\, n))) = \\
\quad \mathsf{Id}(\mathsf{set}(\mathsf{cf}(E, E')))
\end{array}$$

by employing the lub-implies-universal theorem and the chain and commutivity results presented above. (It is left to the reader to generalise the universality situation presented in Figure 2 to two chains instead of one.)

5.3 Inverse limits

Finally, we shall state the main result of this section which follows from the universality of the functor for the continuous function space, i.e. the universality of cf with the family $\mathsf{cf\_emb}$. The result states that this functor preserves inverse limits:

$$\begin{array}{l}\forall \mathbf{e}\mathbf{e}'\mathbf{D}\mathbf{D}'.\\ \quad \mathsf{emb\_chain}\,\mathbf{D}\,\mathbf{e} \Rightarrow \mathsf{emb\_chain}\,\mathbf{D}'\,\mathbf{e}' \Rightarrow\\ \quad \mathsf{Dinf}(\lambda n.\,\mathsf{cf}(\mathbf{D}\,n, \mathbf{D}'\,n))(\lambda n.\,\mathsf{cf\_emb}(\mathbf{D}\,n, \mathbf{D}'\,n, \mathbf{D}(\mathsf{SUC}\,n))(\mathbf{e}\,n, \mathbf{e}'\,n)) \cong\\ \quad \mathsf{cf}(\mathsf{Dinf}\,\mathbf{D}\,\mathbf{e}, \mathsf{Dinf}\,\mathbf{D}'\,\mathbf{e}')\end{array}$$

In the next section this theorem is used to give a model of the λ-calculus.

In order to prove the theorem we do not only use the universality of the continuous function space but the universality of Dinf as well. Since the function space yields a chain of cpos and satisfies commutivity we are able to use the universality of Dinf to obtain an embedding from Dinf to the function space, and vice versa, since Dinf yields a chain of cpos and satisfies commutivity we can use the universality of the function space to obtain an embedding in the other direction. Though this is not enough to establish the isomorphism result (see Exercise 3 of [Plo83]) we are able to prove that the embeddings returned by the universality theorems are each others inverses. A main step of the proof of this is to use the continuity of composition, i.e. the fact that it preserves lubs of chains.

6 D_∞—a model of the λ-calculus

Until this point we have worked with assumed embedding projection chains of cpos and most constants and theorems have been parametrised with such assumptions. We shall now construct one specific but still parametrised chain by iterating the continuous function space construction starting at any cpo D with an embedding $e : D \lhd \mathsf{cf}(D, D)$. From this chain we obtain a partially concrete instantiation of Dinf which, due to the fact that the functor for the function space preserves inverse limits, yields a parametrised model of the λ-calculus (obviously, the parameters are D and e). Choosing one concrete starting point, e.g. the "bottom" cpo void and a certain embedding

$\mathsf{void\_emb}$ which is equal to bottom for all arguments, we obtain a concrete non-trivial (i.e. non-empty) model D_∞ of the λ-calculus.

Just as in the previous section, everything that is done in this section for the function space could have been done for any recursive domain equation $D \cong \mathcal{F}(D)$ where $\mathcal{F}$ is a continuous covariant functor. Starting from the "bottom" cpo void, for convenience written as $\perp\!\!\!\perp$, and from the everywhere undefined embedding $\varepsilon_\perp : D \lhd \mathcal{F}(D)$, the chain

$$\perp\!\!\!\perp \overset{\varepsilon_\perp}{\lhd} \mathcal{F}(\perp\!\!\!\perp) \overset{\mathcal{F}(\varepsilon_\perp)}{\lhd} \mathcal{F}^2(\perp\!\!\!\perp) \overset{\mathcal{F}^2(\varepsilon_\perp)}{\lhd} \cdots$$

which is formed by iterating $\mathcal{F}$ has an inverse limit (usually called D_∞). Note that this construction of a solution to a recursive domain equation resembles the fixed point theorem.

6.1 Constructing a parametrised model

We shall define a sequence of cpos and embeddings by iterating the cf construction on cpos and the $\mathsf{cf\_emb}$ construction on embeddings using any cpo as the initial cpo element and any embedding of this cpo into its own function space as the first embedding element of the sequences. The sequence of cpos is defined by a primitive recursive definition

$$\begin{aligned}&\mathsf{iter\_cf}\,D\,0 \overset{\text{def}}{=} D \land\\&\mathsf{iter\_cf}\,D(\mathsf{SUC}\,n) \overset{\text{def}}{=} \mathsf{cf}(\mathsf{iter\_cf}\,D\,n, \mathsf{iter\_cf}\,D\,n)\end{aligned}$$

and the sequence of embeddings is defined in a similar way:

$$\begin{aligned}&\mathsf{iter\_cf\_emb}\,D\,e\,0 \overset{\text{def}}{=} e \land\\&\mathsf{iter\_cf\_emb}\,D\,e(\mathsf{SUC}\,n) \overset{\text{def}}{=}\\&\quad \mathsf{cf\_emb}(\mathsf{iter\_cf}\,D\,n, \mathsf{iter\_cf}\,D\,n, \mathsf{iter\_cf}\,D(\mathsf{SUC}\,n))\\&\qquad\quad (\mathsf{iter\_cf\_emb}\,D\,e\,n, \mathsf{iter\_cf\_emb}\,D\,e\,n).\end{aligned}$$

Using induction on the natural numbers, it is not difficult to prove that these definitions yield a chain of cpos

$$\begin{aligned}&\forall De.\\&\quad \mathsf{cpo}\,D \Rightarrow \mathsf{emb}(D, \mathsf{cf}(D,D))e \Rightarrow\\&\quad \mathsf{emb\_chain}(\mathsf{iter\_cf}\,D)(\mathsf{iter\_cf\_emb}\,D\,e),\end{aligned}$$

provided the parameters enjoy the simple properties just mentioned informally above.

So, we have constructed a chain of cpos which we can match against both assumptions of the theorem stating that the functor for the continuous function space preserves inverse limits. This yields the following theorem:

$$\begin{array}{l}
\mathsf{cpo}\, D, \mathsf{emb}(D, \mathsf{cf}(D,D))e \\
\vdash \mathsf{Dinf} \\
\quad (\lambda n.\, \mathsf{cf}(\mathsf{iter_cf}\, D\, n, \mathsf{iter_cf}\, D\, n)) \\
\quad (\lambda n.\, \mathsf{cf_emb}(\mathsf{iter_cf}\, D\, n, \mathsf{iter_cf}\, D\, n, \mathsf{iter_cf}\, D(\mathsf{SUC}\, n)) \\
\qquad\qquad (\mathsf{iter_cf_emb}\, D\, e\, n, \mathsf{iter_cf_emb}\, D\, e\, n)) \cong \\
\quad \mathsf{cf}(\mathsf{Dinf}(\mathsf{iter_cf}\, D)(\mathsf{iter_cf_emb}\, D\, e), \mathsf{Dinf}(\mathsf{iter_cf}\, D)(\mathsf{iter_cf_emb}\, D\, e)).
\end{array}$$

The right-hand side of this theorem has the desired form. Note that we have assumed a cpo D and an embedding $e : D \lhd \mathsf{cf}(D,D)$. Now, rewriting the left-hand side of the theorem by folding the definitions of iter_cf and iter_cf_emb we obtain

$$\begin{array}{l}
\mathsf{cpo}\, D, \mathsf{emb}(D, \mathsf{cf}(D,D))e \\
\vdash \mathsf{Dinf}(\lambda n.\, \mathsf{iter_cf}\, D(\mathsf{SUC}\, n))(\lambda n.\, \mathsf{iter_cf_emb}\, D\, e(\mathsf{SUC}\, n)) \cong \\
\quad \mathsf{cf}(\mathsf{Dinf}(\mathsf{iter_cf}\, D)(\mathsf{iter_cf_emb}\, D\, e), \mathsf{Dinf}(\mathsf{iter_cf}\, D)(\mathsf{iter_cf_emb}\, D\, e)).
\end{array}$$

The next step is to use that the inverse limit of a chain is unaffected if we take a suffix of the chain, and as a special case, if we ignore the first element of the chain:

$$\forall \mathbf{D}\mathbf{e}.\, \mathsf{emb\_chain}\, \mathbf{D}\, \mathbf{e} \Rightarrow \mathsf{Dinf}\, \mathbf{D}\, \mathbf{e} \cong \mathsf{Dinf}(\lambda n.\, \mathbf{D}(\mathsf{SUC}\, n))(\lambda n.\, \mathbf{e}(\mathsf{SUC}\, n)).$$

Instantiating this with the chain of cpos derived above and using transitivity of $\cong$ we obtain

$$\begin{array}{l}
\mathsf{cpo}\, D, \mathsf{emb}(D, \mathsf{cf}(D,D))e \\
\vdash \mathsf{Dinf}(\mathsf{iter_cf}\, D)(\mathsf{iter_cf_emb}\, D\, e) \cong \\
\quad \mathsf{cf}(\mathsf{Dinf}(\mathsf{iter_cf}\, D)(\mathsf{iter_cf_emb}\, D\, e), \mathsf{Dinf}(\mathsf{iter_cf}\, D)(\mathsf{iter_cf_emb}\, D\, e))
\end{array}$$

which is the desired result. However, let us define a constant Dinf_cf to abbreviate the Dinf term above

$$\mathsf{Dinf\_cf}\, D\, e \stackrel{\mathrm{def}}{=} \mathsf{Dinf}(\mathsf{iter\_cf}\, D)(\mathsf{iter\_cf\_emb}\, D\, e)$$

and use this to simplify the appearance of the previous result. Let us also discharge the assumptions and generalise the variables, obtaining a theorem which gives *a parametrised model of the λ-calculus:*

$$
\begin{aligned}
&\forall De.\\
&\quad \mathsf{cpo}D \Rightarrow \mathsf{emb}(D, \mathsf{cf}(D, D))e \Rightarrow\\
&\quad \mathsf{Dinf\_cf}\, D\, e \cong \mathsf{cf}(\mathsf{Dinf\_cf}\, D\, e, \mathsf{Dinf\_cf}\, D\, e).
\end{aligned}
$$

6.2 A concrete model derived using void

In order to illustrate how the previous result can be used to derive a concrete model of the λ-calculus, we shall consider a very simple yet non-empty cpo, called void:

$$\mathsf{void} \stackrel{\text{def}}{=} \{\{\}\}, \lambda xy.\, x = y.$$

It contains just one element, the empty set, which trivially is a bottom element of the cpo:

$$\mathsf{bot}\,\mathsf{void} = \{\}.$$

Furthermore, it is easy to define a function void_emb

$$\mathsf{void\_emb}\, D \stackrel{\text{def}}{=} \lambda x \in \mathsf{set}\,\mathsf{void}.\, \mathsf{bot}\, D$$

which embeds void into any pointed cpo D:

$$\forall D.\ \mathsf{pcpo}\, D \Rightarrow \mathsf{emb}(\mathsf{void}, D)(\mathsf{void\_emb}\, D).$$

This is the everywhere undefined embedding. Of course, the associated projection maps all elements of D to the empty set. The last theorem can be used to instantiate the parametrised model if we observe that cf(void, void) is pointed because void is pointed. Hence, defining a constant D_∞ to abbreviate the Dinf_cf term of the instantiated model

$$D_\infty \stackrel{\text{def}}{=} \mathsf{Dinf\_cf}(\mathsf{void}, \mathsf{void\_emb}(\mathsf{cf}(\mathsf{void}, \mathsf{void})))$$

the desired concrete and *non-trivial model of the λ-calculus* can be stated as follows:

$$D_\infty \cong \mathsf{cf}(D_\infty, D_\infty).$$

7 Concluding remarks

Combining set theory and higher order logic (simple type theory) in the same theorem prover provides a useful system for doing mathematics. The simplicity and convenience of higher order logic can be exploited as well as the expressive power of set theory. This paper has demonstrated this by presenting a formalisation of a model of the λ-calculus in HOL-ST. This was done via the inverse limit construction which is not possible in higher order logic. However, rather than working in set theory only it was shown that set and type theoretic reasoning can be mixed to advantage by exploiting set theory only to the extent of which it is necessary and working in higher order logic the rest of the time.

One of the main disadvantages of set theory is the presence of explicit type assumptions. This means that type checking is done late by theorem proving whereas in higher order logic type checking is done early in ML. Furthermore, type checking is automatic in HOL but cannot be fully automated in set theory and was done manually in the present development.

The paper has discussed the issue of whether to use higher order logic or set theory when both approaches are possible. A main problem was to decide when to use the *type* of natural number *num* and when to use the *set* of natural numbers Num to represent certain sequences. Of course, due to the benefits of type checking it would be desirable to use *num* only but this was not possible. The set Num had to be used in the dependent product. The situation was further complicated by the fact that when the decision had to be made we were already using *num* for related things. To make a long story short, we tried to use both *num* and Num for the entire development from that stage and the funny thing is that though the first definitions and theorems were simpler using Num it was much simpler to use *num* in the long run. Hence, the paper presented this approach. A third alternative would be to use Num only, i.e. to change the initial definitions where *num* was used, but this would complicate the entire development with type assumptions occurring everywhere. It is difficult to say which approach to choose in the general situation but I feel that the benefits of type checking should not be underestimated.

The inverse limit construction is only one method of solving recursive domain equations. Another is via universal domains like $\mathrm{P}\omega$ in which domains are encoded as retracts [Sco76, Sto77, Bar84]. A third technique is information systems providing a representation of domains for which equations are solved by the fixed point method [Sco82, LW91, Win93]. There is also some recent work by Pitts [Pit93, Pit94]. Both $\mathrm{P}\omega$ and information systems could probably be formalised in HOL, in fact, Petersen formalised $\mathrm{P}\omega$ in HOL [Pet93] proving that it is a reflexive cpo and hence provides a model of the λ-calculus. However, both techniques would construct separate worlds where domains and HOL types do not share any elements. This would introduce the inconvenience of having translation functions between the two worlds. Though set theory suffers from the same problem its advantage is that it is more general so translation functions could probably be defined for more general use.

Bernhard Reus is doing similar work in the LEGO system which supports a very strong intuitionistic type theory (ECC) with dependent families. This is work in progress which has not been reported anywhere. He has formalised the inverse limit construction of synthetic domain theory [RS94] but has not continued from this to solve recursive domain equations (he plans to consider streams).

Acknowledgements

I would like to thank Mike Gordon for encouragements and discussions. This work is supported by EPSRC grant GR/G23654.

References

[Age93] S. Agerholm, 'Domain Theory in HOL'. In *Proceedings of the 6th International Workshop on Higher Order Logic Theorem Proving and its Applications*, Jeffrey J. Joyce and Carl-Johan H. Seger (Eds.), Vancouver, B.C., Canada, August 1993, LNCS 780, Springer-Verlag 1994.

[Age94a] S. Agerholm, 'LCF Examples in HOL'. In *Proceedings of the 7th International Workshop on Higher Order Logic Theorem Proving and its Applications*, Thomas F. Melham and Juanito Camilleri (Eds.), Malta, September 1994, LNCS 859, Springer-Verlag 1994.

[Age94b] S. Agerholm, *A HOL Basis for Reasoning about Functional Programs.* Submitted for the PhD degree at Aarhus University, June, 1994.

[Bar84] H.P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics.* North Holland, 1984.

[Gor94] M.J.C. Gordon, 'Merging HOL with Set Theory: preliminary experiments'. Technical Report no. 353, University of Cambridge, Computer Laboratory, 1994.

[GM93] M.J.C. Gordon and T.F. Melham (Eds.), *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.

[LW91] K.G. Larsen and G. Winskel, 'Using Information Systems to Solve Recursive Domain Equations'. *Information and Computation*, Vol. 91, No. 2, April 1991.

[Pau87] L.C. Paulson, *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge Tracts in Theoretical Computing 2, Cambridge University Press, 1987.

[Pet93] K.D. Petersen, 'Graph Model of LAMBDA in Higher Order Logic'. In the *Proceedings of the 6th International Workshop on Higher Order Logic Theorem Proving and its Applications*, Jeffrey J. Joyce and Carl-Johan H. Seger (Eds.), Vancouver, B.C., Canada, August 1993, LNCS 780, Springer-Verlag 1994.

[Pit93] A.M. Pitts, 'Relational Properties of Recursively Defined Domains'. In *8th Annual Symposium on Logic in Computer Science*, IEEE Computer Society Press, 1993.

[Pit94] A.M. Pitts, 'A Co-induction Principle for Recursively Defined Domains'. *Theoretical Computer Science*, Vol. 124, 1994.

[Plo83] G. Plotkin, *Domains*. Course notes, Department of Computer Science, University of Edinburgh, 1983.

[RS94] B. Reus and T. Streicher, 'Naive Synthetic Domain Theory—A Logical Approach'. Draft, 1994.

[Sco76] D.S. Scott, 'Data Types as Lattices'. *SIAM J. Comput.*, Vol. 5, No. 3, September 1976.

[Sco82] D.S. Scott, 'Domains for Denotational Semantics'. In *Proc. of ICALP'82*, LNCS 140, 1982.

[SP82] M. Smyth and G.D. Plotkin, 'The Category-theoretic Solution of Recursive Domain Equations'. *SIAM Journal of Computing*, Vol. 11, 1982.

[Sto77] J.E. Stoy, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. The MIT Press, 1977.

[Win93] G. Winskel, *The Formal Semantics of Programming Languages*. The MIT Press, 1993.